Date:

WEEK 2 : Neural Networks Basics

→ Binary classification:

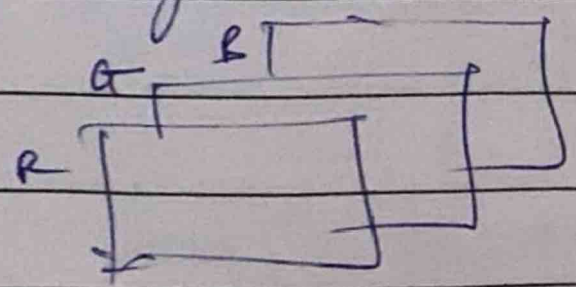
Logistic Regression is an algorithm for binary classification.

Example:

(Input) [Image ~~of a cat~~] → 1 (cat) v/s 0 (Non-cat)

x → y

Image:



3 matrices of 64 x 64 dimension

[matrix of pixels]

Date:

We unroll all these pixel intensity values into a feature vector x

$x = \begin{bmatrix} 126 \\ 25 \\ \vdots \\ 255 \\ 312 \\ \vdots \\ 366 \end{bmatrix}$

64 x 64 x 3
= 12288

Dimension of input feature vector = $n = n_x$
= 12288

$x \longrightarrow y$

Notation :

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples : $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ ... $(x^{(m)}, y^{(m)})$

M_{train} $\quad$ M_{test}

training set $\quad$ test set

Date:

→ Input matrix

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \uparrow n_x \downarrow$$

$\leftarrow m \rightarrow$

$$X \in \mathbb{R}^{n_x \times m}$$

$$X.shape = (n_x, m)$$

→ output matrix

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$$Y \in \mathbb{R}^{1 \times m}$$

$$Y.shape = (1, m)$$

Logistic Regression

Date:

Given x, want $\hat{y} = P(y=1|x)$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = w^T x + b$

For this algo formula, for y being 1
$\hat{y}$ has to be $0 \leq \hat{y} \leq 1$.

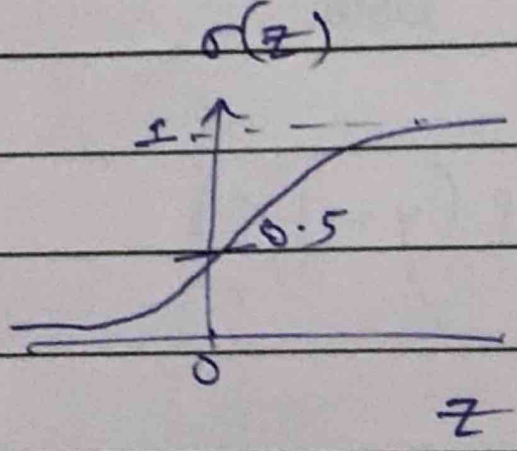
$\because$ '$w^T x$' can be $>>> 1$

(or) it can even be negative.

$\therefore \hat{y} = \sigma(w^T x + b)$

↓ sigmoid function

Date:



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

If z is very large

$$\therefore \sigma(z) \approx \frac{1}{0+1} = 1$$

If z is large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx 0$$

Date:

Logistic Regression Cost function :

$\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

Given $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$

want $\hat{y}^{(i)} \approx y^{(i)}$

Loss (Error) function :

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$ (Not used)

This formula causes optimisation problem.

→ Loss-function is defined for a single training example

Date:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$

$\therefore -\log \hat{y}$ should be ↓

$\therefore \log \hat{y}$ ↑

$\therefore \hat{y}$ ↑.

If $y = 0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$

$\therefore -\log(1-\hat{y})$ ↓

$\therefore \log(1-\hat{y})$ ↑

$\therefore \hat{y}$ ↓

→ Cost function measures over an entire training set.

Date:

Cost function: $J(w,b)$

$$= \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$= \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$

Gradient Desent:

Recap:

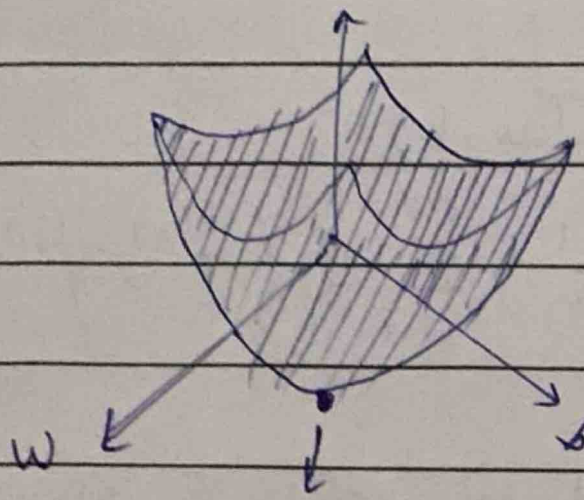
$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$= \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$$

Want to find w, b so that $J(w,b)$ is minimum.

Date:



minimum of $J(w,b)$

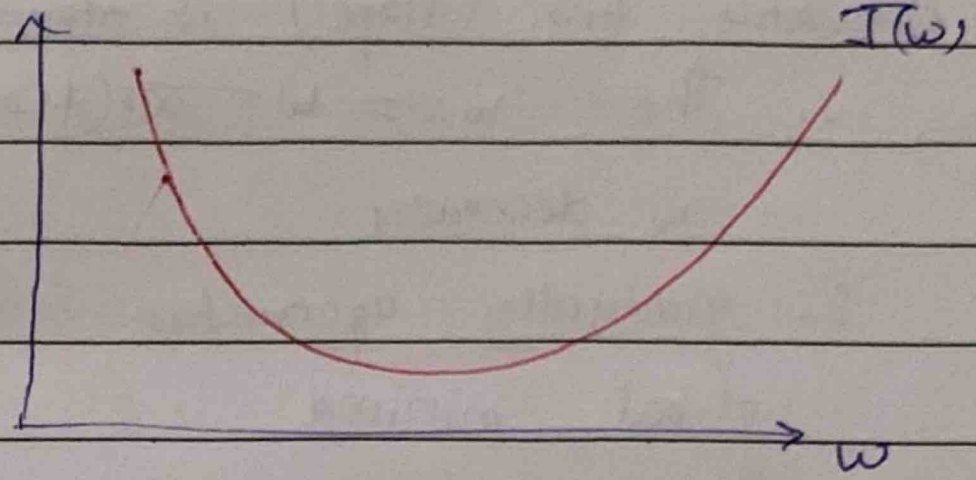
$J(w,b)$ [Convex function] → main reason to use this function for logistic regression

[non-convex functions]

To find good values of parameters, we initilise values of w and b, most often with values of zero.

Date:

Gradien desent graph :



Repeat {

$$w := w - \alpha \frac{d J(w)}{dw}$$

}

→ α : learning rate (controls how big a step we take on each iteration of gradient descent)

→ $\frac{d J(w)}{dw}$ will be written as "dw"

is the update of the change you want to make to the parameters w.

Date:

on RHS of the graph;
the derivative term (slope) is positive, ∴ the $w := w - \alpha \cdot (+ve)$
w decreases
∴ gradually approaches global minima.

on LHS of the graph;
the derivative term (slope) is negative, ∴ $w := w - \alpha \cdot (-ve)$
∴ -w increases
∴ gradually approaches global minima.

→ For $J(w,b)$

$$w := w - \alpha \frac{dJ(w,b)}{dw} \quad \rightarrow dw \quad \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{dJ(w,b)}{db} \quad \rightarrow db \quad \frac{\partial J(w,b)}{\partial b}$$

Date: 22/07/2024

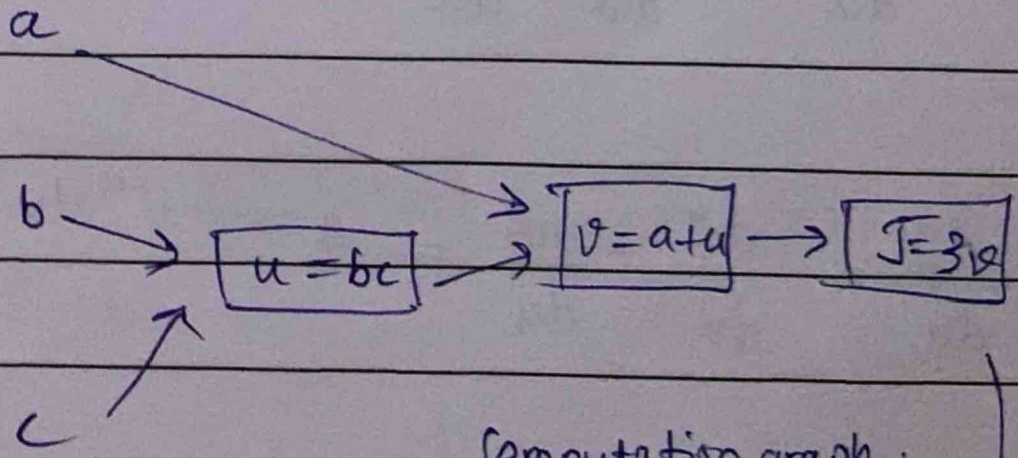
Computation Graph:

example:

$J(a,b,c) = 3(a+bc)$

→ 3 steps:

1. $u = bc$
2. $v = a + u$
3. $J = 3v$



Computation graph:

This is the Final output variable which will be used for optimisation.

Date:

~~For~~ example:

~~If you~~ want to compute derivative of the ~~final (last step)~~ output variable, then we've done one step of backward propagation. ~~Equivalent~~ called one step backward in graph.

$$\frac{dJ}{dv} = 3 \qquad \text{"dv"}$$

$$\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da} = 3 \times 1 = 3 \qquad \text{"da"}$$

$$\frac{dJ}{du} = \frac{dJ}{dv}\cdot\frac{dv}{du} = 3 \qquad \text{"du"}$$

$$\frac{dJ}{db} = \frac{dJ}{dv}\cdot\frac{dv}{du}\cdot\frac{du}{db} = 3c = 3(2) = 6$$

"db"

Date:

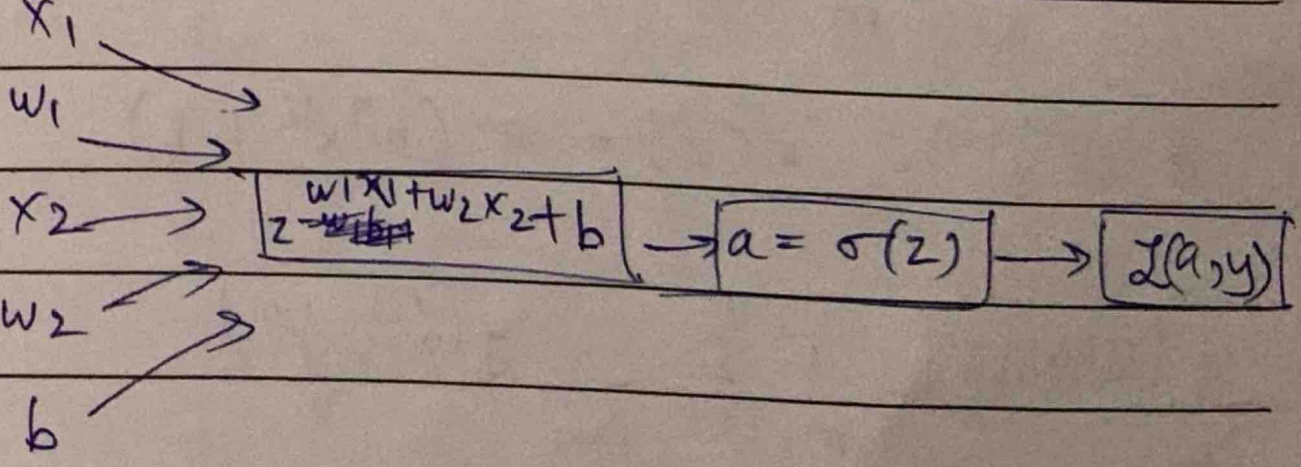
To compute derivatives of all the variables, follow right-to-left computation (backward propagation).

Logistic Regression recap:

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L} = (a, y) = -(y \log(a) + (1-y) \log(1-a))$$



$$\text{"}da\text{"} = \frac{-y}{a} + \frac{(1-y)}{1-a}$$

$$\text{"}dz\text{"} = \text{"}da\text{"} \cdot \frac{da}{dz} \qquad \frac{da}{dz} = a(1-a)$$

$$\therefore \text{"}dz\text{"} = a - y$$

Date:

"dw_1" = $x_1 \cdot dz$

"dw_2" = $x_2 \cdot dz$

"db" = dz

Logistic Regression on 'm' examples:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$\frac{\partial [J(w,b)]}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})$$

{① , ② : for loops}

Date:

$J = 0$, $dw_1 = 0$, $dw_2 = 0$, $db = 0$

① For $i = 1$ to m {

$z^{(i)} = w^T x^{(i)} + b$ $\quad dw = np.zeros((n_x, 1))$

$a^{(i)} = \sigma(z^{(i)})$

$J += -\left[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

② { $dw_1 += x^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$ } assuming $n = 2$ features.

$db += dz^{(i)}$

vectorised: $dw += x^{(i)} dz^{(i)}$

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$

}

Note: Here we are using two for loops making it inefficient, hence vectorisation can be used.

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

Date:

Vectorisation:

→ To get rid of explicit for loops in our code, to make it more efficient.

$$z = w^T x + b$$

$$w \in \mathbb{R}^{n_x}$$

$$w = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x \in \mathbb{R}^{n_x}$$

Non-vectorised:

```
z = 0
for i in range(n-x):{
    z += w[i] * x[i] }
z += b
```

Date:

Vectorised:

$$z = \underbrace{np.dot(w, x)}_{w^T \cdot x} + b$$

GPUs } use SIMD - single instruction multiple data
CPUs }

to process in-built functions of python.

Vectors and matrix valued functions:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \longrightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

Non-vectorised:

```
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i]);
```

vectorised:

```
import numpy as np
u = np.exp(v)
np.log(v)
np.abs(v)
np.maximum(v,0)
```

Date:

Vectorizing Logistic Regression :

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(m)} = w^T x^{(m)} + b$

....

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(m)} = \sigma(z^{(m)})$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad (n_x, m)$$

$X \in \mathbb{R}^{n_x \times m}$

$Z = [z^{(1)}, z^{(2)}, \ldots, z^{(m)}] = w^T X + [b \; b \; b \; \ldots \; b]$

$(1 \times n_x)(n_x \times m) = 1 \times m \qquad 1 \times m$

$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \ldots \quad w^T x^{(m)} + b]$

$z^{(1)} \quad z^{(2)} \quad z^{(m)} \qquad 1 \times m$

$Z = np.dot(w.T, x) + b$

$A = [a^{(1)} \; a^{(2)} \ldots a^{(m)}] = \sigma(Z)$

Date:

$dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$

— —

$dZ = [dz^{(1)}\ dz^{(2)}\ \dots\ dz^{(m)}]$

$1 \times m$

$A = [a^{(1)}\ a^{(2)}\ \dots\ a^{(m)}]$

$Y = [y^{(1)}\ y^{(2)}\ \dots\ y^{(m)}]$

$dZ = A - Y = [a^{(1)} - y^{(1)}\ a^{(2)} - y^{(2)}\ \dots\ a^{(m)} - y^{(m)}]$

$dw = 0$	$db = 0$
$dw += x^{(1)} dz^{(1)}$	$db += dz^{(1)}$
$dw += x^{(2)} dz^{(2)}$	$db += dz^{(2)}$
}	} $db += dz^{(m)}$
$dw /= m$	$db /= m$

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} [np.sum(dz)]$$

$$dw = \frac{1}{m} (X \cdot dz^T)$$

Date:

$$Z = w^T \cdot X + b$$

$$= np.dot(w.T, X) + b$$

$$A = \sigma(Z)$$

$$dZ = A - Y$$

$$dw = \frac{1}{m}\left[(X \cdot dZ^T)\right]$$

$$db = \frac{1}{m}\left[np.sum(dZ)\right]$$

$$w := w - \alpha\, dw$$

$$b := b - \alpha\, db$$

Date:

Broadcasting Example :

	Apples	Beets	Eggs	Potatoes	
Carb	[				] = A
Protein					(3x4)
Fat	[				]

```
cal = A.sum(axis=0)
percentage = 100 * A / cal.reshape(1,4)
```

General Principle :

- (m,n) matrix $\pm$ (1,n) ⟶ (m,n)
- (m,n) matrix $\pm$ (m,1) ⟶ (m,n)

(m,1) + R

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 1 \end{bmatrix} + 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \\ \vdots \\ 1 \end{bmatrix}$$

$$[1\ 2\ 3\ \cdots] + 100 = [101\ 102\ 103\ \cdots]$$